# 8

## Фокус занятия

- Слайс как фрагмент массива. len/cap/range.
- Создание слайса: make/append. Слайс - это адреса!
- Слайсы как параметры функции.

## Небольшое вступление для преподавателей.

Занятие про слайсы. И занятие не очень простое получается. Придётся говорить о том,что слайсы - это некий набор указателей, т.е. разговор будет такой, несколько рискованный, с погружением в технические моменты. С другой стороны

- на указательную сущность слайсов выходим не с самого начала - есть возможность как-то смягчить разговор, даже может быть что-то оставить для повторного возвращения к этой теме, попозже, немного с других позиций
- на прошлом занятии уже был разговор о передаче адреса массива - тут этот разговор получает своё развитие, т.е. есть на что опираться
- и вообще, схожесть массивов и слайсов и их различия - на противопоставлениях одних другим многое можно объяснить довольно наглядно, так чтобы дети реально “пощупали” адресную работу, при этом не марая рук об явное задание адресных типов, вообще даже не приближаясь к этой теме.

## Слайс как отрезок массива.

Слайс - это отрезок массива. Но при этом массив живёт сам по себе, а слайс - это что-то вроде окошка, которым ограничивается часть массива, над которой он кружится. Т.е. слайс не владеет своими данными, слайс - это не данные, это только пара указателей, пара индексов. Слайсов без массива не бывает: сам массив конечно имеет отношение к каждому своему слайсу, но слайсов может быть много, а данные хранятся только в одном экземпляре в соответствующем массиве.

**Объявление слайса**

Запись `arr[from:to]` обозначает отрезок массива arr, начиная с элемента arr[from] и вплоть до элемента arr[to-1] включительно. Удобно в таком достаточно распространённом выборе то, что количество элементов в таком отрезке как раз (from - to).

Начнём с примера `08_0a.go`, в котором использованы три способа объявления переменной, в данном случае - слайса: полное объявление с одновременной инициализацией (слайс b), полное объявление, а присваивание значения происходит позже (слайс c), краткое обявление (слайс d). И в конце примера мы выбираем слайс с нулевого элемента `d = a[:4]` и слайс до последнего элемента массива `d = a[3:]`, показываем, как можно сократить запись в таких случаях (причём выглядит такая запись даже естественнее).

```
package main

import "fmt"

func main() {
    a := [7]int{0, 1, 2, 5, 10, 20, 50}
    var b []int = a[1:4]    //  b - слайс (отрезок) массива a от a[1] до a[3]
    fmt.Println(b)          //  [1 2 5]
    for i:= 0; i<3; i++  {
        b[i] *= i
    }
    fmt.Println(b)          //  [0 2 10]
    fmt.Println(a)          //  [0 0 2 10 10 20 50]
    for i, x:= range a  {
        a[i] = x + 15
    }
    var c []int
    c = a[2:6]              //  c - слайс (отрезок) массива a от a[2] до a[5]
    fmt.Println(a)          //  [15 15 17 25 25 35 65]
    fmt.Println(b)          //  [15 17 25]
    fmt.Println(c)          //  [17 25 25 35]
    for i, x:= range c  {
        c[i] = x - 35
    }
    fmt.Println(a)          //  [15 15 -18 -10 -10 0 65]
```

```
    fmt.Println(b)          //  [15 -18 -10]
    fmt.Println(c)          //  [-18 -10 -10 0]
    d:= a[2:3]
    fmt.Println(d)          //  [-18]
    d = a[:4]
    fmt.Println(d)          //   [15 15 -18 -10]
    d = a[3:]
    fmt.Println(d)          //   [-10 -10 0 65]
}
```

Заодно видим, что обращение к элементам слайса подобно способу обращения к элементам массива. И range для слайсов совершенно аналогичен range для массива.

**Функции len и cap для слайсов**

len - длина (length) слайса - это количество элементов в нём
cap - ёмкость (capacity) слайса - это количество элементов в массиве, над которым создан слайс, начиная с начального элемента слайса и до конца массива.
К слову, для массивов эти функции тоже определены и имеют вполне понятный смысл, так что сходство слайсов и массивов здесь не нарушается. Ну, ничего, дальше будет смешнее...

Соответственно, вот этот код `08_0b.go`

```
package main

import "fmt"

func main() {
    coins := [...]int{1, 2, 5, 10, 20, 50, 100, 200}
    fmt.Println(len(coins), cap(coins), coins) //   8 8 [1 2 5 10 20 50 100 2
00]
    cent := coins[0:6]                         //
    fmt.Println(len(cent), cap(cent), cent)    //   6 8 [1 2 5 10 20 50]
    cent[6] = -1                               //   panic: runtime error: ind
ex out of range
}
```

успевает вывести то, что написано в комментариях, но при попытке обратиться к элементу за границами слайса программа вылетает с таким сообщением об ошибке (runtime error):

```
panic: runtime error: index out of range

goroutine 1 [running]:
main.main()
        H:/go/sandbox/08_0b.go:12 +0x244
exit status 2
```

12-я строка - это как раз `cent[6] = -1` - попытка обратиться к элементу за границами слайса.

Кажется, что функция cap для слайса не имеет смысла, но это не так. Дело в том что слайс можно "переслайсить", базируясь либо на массиве, либо на самом слайсе вплоть до ёмкости базы.
Пример `08_0c.go`

```
package main

import "fmt"

func main() {
    coins := [...]int{1, 2, 5, 10, 20, 50, 100, 200}
    fmt.Println(len(coins), cap(coins), coins) //   8 8 [1 2 5 10 20 50 100 2
00]
    cent := coins[0:6]                         //
    fmt.Println(len(cent), cap(cent), cent)    //   6 8 [1 2 5 10 20 50]
    cent = cent[3:5]                           //
    fmt.Println(len(cent), cap(cent), cent)    //   2 5 [10 20]
    cent = cent[0:5]                           //
    fmt.Println(len(cent), cap(cent), cent)    //   5 5 [10 20 50 100 200]
    cent = cent[3:5]                           //
    fmt.Println(len(cent), cap(cent), cent)    //   2 2 [100 200]
    cent = cent[0:3]                           //
}
```

как и предыдущий, успевает вывести всё то, что написано в комментариях,

после чего слетает с runtime error на операторе `cent = cent[0:3]` :

```
panic: runtime error: slice bounds out of range

goroutine 1 [running]:
main.main()
        H:/go/sandbox/08_0c.go:18 +0x55a
exit status 2
```

## Внутренняя структура слайса. Передача слайса в функцию-1.

А как устроен слайс внутри? Поскольку слайс - это надстройка над каким-то существующим массивом, то хранить повторно элементы самого массива нет смысла. Чтобы успешно работать со слайсом, нужно знать, где находится первый элемент слайса, остальные элементы мы уж как-нибудь найдём (также, как это происхордит в массиве). А, нет, мы ещё также можем проколоться, если

- залезем за правый край слайса - т.е. надо знать, где находится последний элемент слайса, или, ещё проще, его длину; да, len слайса тоже где-то хранится
- переслайсим мимо массива, к которому привязан слайс - заметить это позволяет знание ёмкости слайса; т.е. cap слайса тоже хранится.

И это всё. Каким бы громадным массив ни был, любой слайс (в смысле переменная типа слайс) хранит в себе в том или ином виде место первого элемента, длину слайса и его ёмкость. На самом деле именно это и хранится, хотя можно было хранить место (адрес, конечно, но можно в данном месте избежать употребления этого слова, хотя особого смысла в таком избегании я не вижу) первого элемента слайса, адрес последнего в слайсе вместо len и адрес последнего в опорном (реальном) массиве вместо cap.

А что же произойдёт, когда мы передадим в функцию слайс? Да именно то, что произойдёт с массивом, когда мы передадим в функцию его адрес, как это было показано в примере `07_4b.go` с прошлого занятия. Да, при передаче слайса, его опорный массив остаётся на месте, а передаём мы вышеперечисленные адреса. Собственно, для обращения к элементам слайса важен только адрес начального элемента слайса, а len и cap исполняют только контрлольные функции. И важно то, что теперь получается так, что изменения слайса изменяют его массив (подобно тому, как в примере `08_0a.go)` ,

который остался в вызывающей функции, т.е. вот такая вот передача по адресу получается.

Что происходит в результате - в примерах `08_1a.go` и `08_1b.go`.

Пока всё хорошо и удобно, просто и понятно (ну, надо надеяться, что серьёзных проблем на этом месте не возникнет). Слайс - это просто фрагмент массива, этакое “окошко”, которое даёт нам доступ только к некоторому отрезку массива, причём хранит в себе только некоторые реперные точки - важные адреса, и поэтому обеспечивает доступ к элементам этого отрезка вот такой, как мы только что рассмотрели.

## Создание слайса без привязки к именованному массиву.

А вот ещё есть возможность создать слайс, который не привязан к к какому-то объявленному массиву. Ну, то есть массив, который последовательно расположенные однотипные данные, создаётся при создании слайса, но доступиться к элементам этого массива можно только через слайс прямо или косвенно (т.е. через слайсы, созданные на базе другого слайса). Например `08_2a.go`:

```
package main

import "fmt"

func main() {
    a := []int{3, 24, 17, 93, 31, 44, 60}
    fmt.Println(len(a), cap(a), a) //   7 7 [3 24 17 93 31 44 60]
    b := a[2:5]                    //
    fmt.Println(len(b), cap(b), b) //   3 5 [17 93 31]
}
```

Создаётся 7-элементный массив и слайс `a` над ним. После “переслайсивания” `a` доступиться до первых дух элементов массива - 3 и 24 - уже невозможно, хотя они по-прежнему хранятся в памяти. Так что, видимо, лучше было бы завести другой слайс: `b := a[2:5]`

### Создание слайса с помощью функции make

Функция make([]T, len, cap) []T создаёт слайс типа []T длины len и ёмкости cap - пример `08_2b.go` :

```
package main

import "fmt"

func main() {
    a := make([]int, 3, 7)          //
    fmt.Println(len(a), cap(a), a)  //   3 7 [0 0 0]
    var b []int = make([]int, 5, 7) //
    fmt.Println(len(b), cap(b), b)  //   5 7 [0 0 0 0 0]
    c := a[:6]                      //
    fmt.Println(len(c), cap(c), c)  //   6 7 [0 0 0 0 0 0]
    d := make([]int, 7)             //
    fmt.Println(len(d), cap(d), d)  //   7 7 [0 0 0 0 0 0 0]
}
```

Получается довольно забавный эффект - почти массив, но только его размер мы задаём уже во время выполнения программы. Программы становятся гибче. Большой пример `08_3.go` как раз и использует эту возможность. И выглядит всё довольно приятно, и массив нужного размера получается, хоть он и слайс. В общем сплошная радость в доме.

Пара замечаний к примеру *`08_3.go`*. В нём вычисляются значения полинома a[0] + a[1] `*` x + a[2] `*` $x^2$ + ... + a[n] `*` $x^n$ для значений x, изменяющихся от начального значения x0 до конечного значения xk с шагом h. Все эти величины вводит пользователь.

1. Да конечно, сюжет там математизированный - вычисление значений полинома; а что делать, надо терпеть - ведь у нас пока только числовые данные.
2. Вариант с вычислением значения полинома по схеме Горнера требует некоторых, хотя и не чрезмерных, но усилий для понимания механизма его работы. Можно этот вариант пропустить - пафос примера вовсе не в нём.

## Расширение слайса - функция append

А вот теперь выкатываемся на скользкий лёд.

Массив - штука стационарная, изменить его размер во время выполнения программы нельзя. А вот слайсы - они как раз динамические и их размер можно изменять, мы это видели, было “переслайсивание”, было присваивание слайсу другого фрагмента массива.
Но есть ещё и возможность добавить в слайс новые элементы с помощью функции `append`. Первый аргумент функции `append` - слайс. А вот дальше могут идти

- один элемент, тип которого есть базовый тип слайса (то же, что и и у элементов слайса)
- несколько элементов того же типа, перечисленные через запятую
- слайс с элементами того же типа, вслед за которым надо поставить три точки - без объяснений, просто “так надо”, не хватало нам ещё тут про variadic заговорить...

Пример `08_4a.go` показывает, как работают все три варианта:

```
package main

import "fmt"

func main() {
    a := [...]int{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}
    s := a[:5]
    fmt.Println(a) //   [1 2 3 4 5 6 7 8 9 10 11 12 13 14]
    fmt.Println(s) //   [1 2 3 4 5]
    s = append(s, 21)
    fmt.Println(a) //   [1 2 3 4 5 21 7 8 9 10 11 12 13 14]
    fmt.Println(s) //   [1 2 3 4 5 21]
    s = append(s, 41, 42, 43)
    fmt.Println(a) //   [1 2 3 4 5 21 41 42 43 10 11 12 13 14]
    fmt.Println(s) //   [1 2 3 4 5 21 41 42 43]
    s = append(s, a[12:]...)
    fmt.Println(a) //   [1 2 3 4 5 21 41 42 43 13 14 12 13 14]
    fmt.Println(s) //   [1 2 3 4 5 21 41 42 43 13 14]
}
```

Пример `08_5.go` - пример с сюжетом - копия примера `07_2.go`, в котором мы строим последовательность $(a^n)\%100$ и ищем её период. Отличие здесь одно - вместо массива с заранее определённой длиной (легко понять, что

последовательность начнёт повторяться не поздее 51-го элемента: в самом деле, все числа не превосходят 99, и все числа имеют одинаковую чётность - такую же, как и число a) мы делаем вид, что мы не знаем, какого размера массив нам нужен (ну, мы действительно не знаем - 51 точно хватит, но вполне себе может оказаться, что достаточно гораздо меньше). С точки зрения слайсов здесь интересно то, что слайс `x` не привязан к именованному массиву. Тем не менее, массив, в котором хранятся данные, существует, но об этом мы уже говорили.

Но давайте посмотрим на пример `08_4b.go`:

```
package main

import "fmt"

func main() {
    a := [...]int{1, 2, 3, 4, 5, 6, 7, 8}
    s := a[:6]
    s2 := s
    fmt.Println(len(a), cap(a), a)    //   8 8 [1 2 3 4 5 6 7 8]
    fmt.Println(len(s), cap(s), s)    //   6 8 [1 2 3 4 5 6]
    fmt.Println(len(s2), cap(s2), s2) //   6 8 [1 2 3 4 5 6]
    s = append(s, -1, -2, -3, -4)     //
    fmt.Println(len(a), cap(a), a)    //   8 8 [1 2 3 4 5 6 7 8]
    fmt.Println(len(s), cap(s), s)    //   10 16 [1 2 3 4 5 6 -1 -2 -3 -4]
    s2 = append(s2, 0)                //
    fmt.Println(len(a), cap(a), a)    //   8 8 [1 2 3 4 5 6 0 8]
    fmt.Println(len(s2), cap(s2), s2) //   7 8 [1 2 3 4 5 6 0]
    s2 = append(s2, -11, 12, 13, 14, 15)
    fmt.Println(len(a), cap(a), a)    //   8 8 [1 2 3 4 5 6 0 8]
    fmt.Println(len(s2), cap(s2), s2) //   12 16 [1 2 3 4 5 6 0 -11 12 13 14
15]
}
```

И вот тут-то начинается ужас-ужас-ужас...
И объясняем детям, что:

- если мы хотим добавить что-то к слайсу, не выходя за пределы его ёмкости, то все изменения отражаются на текущем массиве, на котором базируется слайс; и всё хорошо, просто и понятно

- а вот если мы хотим добавить что-то в слайс так, что он уже не будет вмещаться в базовый массив, то базовый массив оставляем в покое и создаём новый, неименованный массив, на котором будет базироваться слайс.
    - новый массив имеет ёмкость в два раза большую, чем слайс до начала добавления
    - копируем в него весь старый базовый массив, который мы таки окончательно оставляем после этого в покое
    - слайс теперь базируется на новом массиве, и мы добавляем в него всё то, что хотели добавить
    - если удвоения окажется недостаточно, то новый массив всё равно будет иметь достаточный размер, только я до конца не понял какой именно; кажется, что ёмкость слайса (т.е. размер базового массива) в таком случае станет минимальным чётным числом, которое позволяет вместить расширенный слайс (если новая длина слайса чётная, то ёмкость будет совпадать с длиной, если нечётная - то на 1 больше); в общем, если детям станет интересно - пусть сами исследуют вопрос, но лучше на этом не заостряться вообще, как по мне, так я бы уклонился от этого вопроса, если дети сами не спросят.

## Передача слайса в функцию-2.

Итак, слайс - это набор адресов, Какие неожиданные эффекты при этом появляются мы только что видели. А что же происходит, когда мы передаём слайс в функцию? Разберёмся поподробнее с учётом новых представлений.

Первое, что мы уже видели и оценили, - это то, что при передаче слайса в функцию все его изменения внутри функции отражаются на базовом массиве, который остался жить в вызывалке. Замечательно. Правда, есть и но. А именно: если мы хотим что-то там делать со слайсом внутри функции, но не хотим, чтобы эти изменения сохранялись в базовом массиве из вызывалки, то у нас ничего не получится. Ну, ладно, можно это стерпеть.
В конце концов, можем сформулировать себе некое не то, чтобы правило, но, скажем так, указание для работы с массивами:

```
хочешь, чтобы изменения параметра-массива в функции отражались на его прообра
зе в вызывалке -
                                                передавай слайс;
хочешь, чтобы не отображались -
                                                передавай массив
```

И тут пара примеров, совсем уж пограничных, если не заграничных: `08_6a.go` и `08_6b.go`. В общем, с ними надо быть поаккуратнее, может быть даже и не трогать их, хотя лучше всё-таки потрогать и дать детям для самостоятельных экспериментов. И совсем уж хорошо было бы, если дети поэкспериментуриуют, соберут свои недоумения и поделятся ими на следующем занятии. Тогда можно будет как раз на этих недоумениях слегка потоптаться и что-то всем вместе понять.

Идея примера `08_6a.go`: если всё-таки передаём слайс, но не хотим чтобы этот отражалось на прообразе, то для переданного слайса создаём копию, над ней и измываемся. Да, непросто, но есть такой эффект, надо его учитывать и использовать.

Плодить внешне парадоксальные примеры в данном случае легко и даже скучновато. Последний, который я себе позволю, пример `08_6b.go` показывает, что изменения ёмкости слайса посредством функции append, не изменяют массив-прообраз (даже если этот массив-прообраз был не именованный). Но, думаю, это вкручивать на лекции совершенно ни к чему. Не надо устраивать из программирования турнир по решению головоломок. Повторюсь. Для домашних исследований - оно неплохо. Ещё лучше, хоть это я и размечтался совсем уж несбыточно, если дети расскажут, что у них получилось. Хотя... Всякое бывает. В общем, хорошо бы, чтобы дети уложили себе это как-то в голове и потестировали свои представления.

Всё. Мечтать не вредно...

**Упражнения** годятся из прошлого занятия, которое про массивы, но можно и какие-то специфически слайсовые. Например

1. Есть слайс. Убрать в нём соседние повторы. [ 1 3 3 2 1 3 5 3 4 4 2 ] –> [ 1 3 2 1 3 5 3 4 2 ]. Очень удобно здесь склеивать два слайса.
2. Та же идея: задача Иосифа - вариант реализации с вычёркиваниями
3. Есть два возрастающих слайса. Склеить их в один слайс, чтобы он тоже возрастал.
4. Всякая длинная арифметика. Да хотя бы увеличить длинное число на 1. Или уменьшить.
5. Наподобие примера с последовательностью: создать слайс с числами Фибоначчи, не превосходящими 1000000. Тут работает идея о том, что слайс - это “массив, который может изменять свой размер”
6. Да вообще, любая рекуррентная последовательность. Например, первые пять чисел даны, а потом x[k] = max (x[i-5], x[i-2]). С какого места эта последовательность станет стабильной (все элементы станут

одинаковыми)? Понятно, что ответ зависит от стартовых значений, а также от параметров 2 и 5 (в смысле, их можно изменить или брать максимум не двух, а трёх, например, чисел)
7. Выкинуть из слайса все чётные числа. Эти числа загнать в другой слайс.

## Комментарии к пройденному.

Занятие довольно необычное. Посмотреть на него - выглядит ужасно сложным и перенасыщенным техническими деталями. Мне тоже было страшновато. Тем не менее, получилось, по ощущениям, неплохо. Это, конесчно, мои ощущения, вскрытие ещё покажет, что там получилось, но мне пока более или менее нравится то, что было.

А было вот что. Я решил, что раз уж технических деталей много, так чего от них бегать. И начал прямо с технических деталей. Т.е. прямо по дорожке, проложенной примерами. Сначала вполне себе естественные примеры 08_0.*go*, на которых несколько раз подчеркнул, что слайсов нет, что они просто некая надстройка над имеющимся реально массивом; а слайс - это эфемер, что он паразитирует на массиве *(*в том смысле, что без массива его нет*)*. А что слайсы имеют адресную природу дети сказали сами. И это было круто. Конечно, они сформулировали не так, но смысл был именно тот. И это сказал не один или два, а из двух разных углов прозвучало “общее мнение группы товарищей, сидящих в этом углу”. Ну, то есть видно было, что большая часть въезжает понемножку. Так что эти мягкие примеры пришлись очень кстати, несмотря на их техничность. И примеры *08_1*.go пролетели как по маслу. - в них же чётко подчёркивается адресная природа слайсов. Да, на слово адрес я особо не нажимал, употреблял также и слова “указатель”, “начало-конец”, возможно ещё какие-то, но и слова “адрес” не избегал. Примеры 08_2*.go мы не разбирали, поскольку темп пошёл хороший, я просто на доске написал вводимые там два способа создания слайса:

```
a := []int{3, 24, 17, 93, 31, 44, 60}    и      a := make([]int, 3, 7)
```

и мы полетели дальше.

И если пример 08_4a.go - вводный пример в серию 08_4*.go - обходится без парадоксов и патологий, то в 08_4b.go и 08_4c.go парадоксы встают в полный рост. Я активно использовал понятие “безымянный массив”, т.е. массив есть, а доступиться к нему напрмяую мы никак не можем. Только через слайсы, которые над ним нависают. Дети меня откровенно радовали, так что может это

мне повезло, но так было. Спросили, а можно ли задвинуться влево от начала слайса (ну, можно ли написать b= a[-2:5], где a - это тоже слайцс, не массив), огорчились, что нельзя. Спросили сами, опередив меня, а что будет если мы приаппендим к слайсу слишком сного элементов, и обычного удвоения ёмкости (cap) не хватит. Ответил, хотя был позыв дать им поковыряться самим, но в это раз язык не повернулся. Тем более по ходу пьесы вот таких упражнений типа "потестируй Go в неоднозначной ситуации и узнай, как будет" уже было несколько штук. А вот пример 08_3.go я с самого начала решил не давать, даже не распечатывал, и оказался прав - он бы очевидно сбил с темпа. Но я файл с примером детям дал - посоветовал попробовать разобраться, но если нет - ну, нет, пусть разберутся с другими примерами - тоже неплохо.

И естественным образом возник вопрос: "так что с этой фигнёй - слайсами - делать, если они такие странные, и их применение так легко приводит к трудноопредсказуемым и сложносознаваемым последствиям?" Я рискнул на обсуждение, хотя времени уже было много. Да, сорри, в итоге лекция длилась минут 80+, но я и сам потерял время - дети выглядели довольно свежими к исходу часа. Причём практически все, и почти все участвовали в обсуждениях, включая подтормаживающих, в общем, просто соловьиный сад... Правда в обсуждении, что делать со слайсами уже пошла усталость, так что я довольно жёстко управлял, но пасьянс более или менее сложился.

По ходу обсуждения пример 08_5.go (который я переименовал сейчас в 08_5b.go) разбирать не стали, я на него в конце обсуждения указал как на пример, аналогичный примеру 08_5a.go. Пример 08_5a.go я набросал прямо на доске (а вот сейчас выкладываю) - заполняет слайс числами Фибоначчи, не превосходящими заданного числа N. Так что с числами Фибоначчи прозвучало, а с последними цифрами последовательности степеней - нет, остался этот пример 08_5b.go для домашних штудий.

И мы сформулировали общие рекомендации. Грубо, конечно, но на то они и рекомендации, а не чёткие правила. Общий смысл был такой:

- если ты используешь слайс для того, чтобы выделять память без запаса, чтобы при необходимости что-то добавить к данным ты хочешь использовать append - пользуйся слайсом, базирующемся на "безымянном массиве"
- если же ты используешь слайс, чтобы манипулировать какими-то фрагментами данных, передавать фрагменты массивов, ещё что-то, но применение append'а не предусматривается - пиши над конкретным массивом, массивом "с именем"

И ещё совсем чуть-чуть поговорили о том, что массив всё-таки побыстрее слайса работает, что перераспределение памяти - не хухры-мухры. И слегка поговорил о том, где использовать слайс над “безымянным массивом”, а где массив с запасом.

И в продолжнеие этой темы я дал на практику нулевое задание: считать и сохранить массив чисел; признак окончаия ввода - введённый `0`. Понятно, что раз размер здесь никак не оговаривается и никаких контекстов нет, то делайте со слайсом, в который всё время прицепляем введённые числа.

Кроме того, дал задания склеить два возрастающих массива в один возрастающий слайс, расщепить слайс на два слайса - чёитные числа и нечётные, всякие заполнения из предыдущего занятия. И пару непростых задач: задачу Иосифа и задачу “сжать слайс на повторы - заменить несколько рядом стоящих одинаковых чисел одним таким числом”.

Может мне слишком повезло, и всё совсем не так радужно (то есть всё, конечно, не так радужно, но вопрос насколько не так), но вот такое построение занятия оказалось удачным: большую часть времени занимаемся техническими вопросами, двигаясь от совсем простых к совсем непростым, а совершенно естественные применения даём по концовке. По-любому, вариант работающий.